Problem:
You find that the initial affine registration calculated by the intensity based registration software is failing. You would like to specify an alternative.

Solution:
Calculate an alternative affine transformation and supply that to the registration software.

Steps:
1. Make a paired landmarks file in Amira using the reference and sample brain
2. Start up the R analysis suite
3. Use the functions in Affine.R to calculate an affine transformation
4. Make a copy of the faulty registration folder and overwrite the **registration** file with one created in R
5. Re-run the registration

Details
Landmark files
See the Amira help Tutorial section "Warping - how to work with landmark sets" for details on how to generate a paired landmark file. I normally specify about 10-15 landmark points for when generating a 9 degrees of freedom registration (translation, rotation, scaling). Save the landmark file in the default ASCII format.

Reading Landmark files
You can use the ReadAmiraLandmarks function to read in landmark pairs.

```
landmarks=ReadAmiraLandmarks("/GD/projects/PN2/analysis/examples/Calculating an Affine Registration from
Landmarks/Template2FemaleAverageGoodBrains.landmarkAscii")
```

The R format is simple, just a 2 element list in which element is a 3 column matrix with n rows.

```
> landmarks
[[1]]
         [,1]    [,2]     [,3]
 [1,] 382.953 249.372  92.9127
 [2,] 372.511 182.524  51.8634
 [3,] 346.202 271.661 108.6400
 [4,] 453.090 298.383  81.8853
 [5,] 443.948 249.159  73.7779
 [6,] 356.750 225.953  79.4531
 [7,] 389.097 273.068  77.8316
 [8,] 396.130 248.455 114.3150
 [9,] 396.833 239.314  60.8060
[10,] 406.678 302.603 112.6940
[11,] 415.819 254.081  90.8036
[12,] 391.207 285.022  90.8036
[13,] 328.318 238.559  99.7218
[14,] 345.502 275.218  89.9928

[[2]]
          [,1]     [,2]    [,3]
 [1,]  74.4207  98.2352 50.5358
 [2,]  47.5778  22.4161 10.0879
 [3,]  35.1298 125.9260 71.9297
 [4,] 139.6640 139.8640  0.0000
 [5,] 134.8600  97.4344  7.0000
 [6,]  45.1979  73.4176 45.0000
 [7,]  74.8186 119.8500 16.0000
 [8,]  99.6359 114.2460 77.0000
 [9,]  76.4197  76.2196  0.0000
[10,]  97.6345 161.8790 57.0000
[11,] 105.2400 108.2420 31.0000
[12,]  79.6220 137.4620 34.0000
[13,]  14.7767  89.4288 65.0000
[14,]  34.3904 130.2570 52.0000
```

so you can make your own if your landmarks are coming from somewhere else.

Calculating a Transformation
This is done using the **CalculateIGSParamsFromLandmarkPairs** function defined in Affine.R. The function calculates a transformation that optimally maps the first set of landmark points onto the second. In this case the first brain is the sample and the second is the reference. However, although this might seem counterintuitive, the warping registration is actually calculated by mapping the reference onto the sample. An affine matrix can be simply inverted so this doesn't matter especially, but in fact if you try and find the optimal landmark registration for the forward and inverse registrations they will be slightly different. Anyway, what I actually wanted to do was to map the second set of points onto the first. I therefore proceeded as follows:

```
> AffineTransform6Dof=CalculateIGSParamsFromLandmarkPairs(landmarks,dofs=6,Swap=T)
Final score =  183.2619
```

```
> AffineTransform9Dof=CalculateIGSParamsFromLandmarkPairs(landmarks,dofs=9,AffineTransform6Dof,Swap=T)
Final score =  101.5640
```

The Swap parameter indicates that I want to map 2->1 instead of 1->2. I have calculated 2 successive transformations, the first with 6 degrees of freedom (translate, rotate and scale) and the second with 9 degrees of freedom, initialised with the 6dof transform. This iterative approach is normally more successful. There are a whole load of a parameters that can be used to change the numerical optimisation technique, change the error terms etc if you look at the function definition.

Manipulating a Transformation
The result is a set of transformation parameters, 15 in all:

```
>AffineTransform9Dof
            [,1]       [,2]       [,3]
[1,] 310.6707991 150.526498 44.9738000
[2,]  11.2416661  -9.915760  1.9623692
[3,]   0.9449208   0.938063  0.6084156
[4,]   0.0000000   0.000000  0.0000000
[5,]  84.3900000  84.390000 43.5000000
attr(,"optresults")
attr(,"optresults")$par
[1] 310.6707991 150.5264982  44.9738000  11.2416661  -9.9157596   1.9623692   0.9449208   0.9380631
0.6084156

attr(,"optresults")$value
[1] 101.5640

attr(,"optresults")$counts
function gradient
     205       29

attr(,"optresults")$convergence
[1] 0

attr(,"optresults")$message
NULL
```

They are (reading off the 5 rows):
translation
rotation (degrees)
scale
shear
centre of rotation

Although this is a convenient representation, it lacks generality because you need to know in what order to apply the transformations. Therefore the most general way to specify the transformation is as a homogeneous affine transformation matrix. This is a 4x4 matrix with last row (0,0,0,1). The top 12 numbers specify the transformation. To find this homogenous affine matrix use:

```
> AffineMatrix9Dof=ComposeAffineFromIGSParams(AffineTransform9Dof)
> AffineMatrix9Dof
            [,1]          [,2]       [,3]      [,4]
[1,]  0.93025970 3.357976e-05 -0.1067603 321.19742
[2,] -0.03187368 9.206038e-01 -0.1150208 164.91997
[3,]  0.16271542 1.801414e-01  0.5878281  33.96959
[4,]  0.00000000 0.000000e+00  0.0000000   1.00000
```

This can then be used to transform points using the convenience function

```
> TransformPoints(landmarks[[2]], AffineMatrix9Dof)
          [,1]     [,2]      [,3]
 [1,] 385.0361 247.1709  93.48158
 [2,] 364.3809 182.8795  51.67925
 [3,] 346.2023 271.4548 104.65254
 [4,] 451.1259 289.2277  81.89037
 [5,] 445.9082 249.5148  77.58016
 [6,] 358.4415 225.8919  81.00180
 [7,] 389.0940 271.0293  77.13893
 [8,] 405.6680 258.0629 116.02509
 [9,] 392.2902 232.6522  60.13456
[10,] 405.9430 304.2782 112.52354
[11,] 415.7920 257.6479  88.81530
[12,] 391.6413 285.0195  91.67407
[13,] 328.0072 239.3011  90.69264
[14,] 347.6423 277.7578  93.59718
>
> landmarks[[1]]
         [,1]    [,2]     [,3]
 [1,] 382.953 249.372  92.9127
 [2,] 372.511 182.524  51.8634
 [3,] 346.202 271.661 108.6400
 [4,] 453.090 298.383  81.8853
 [5,] 443.948 249.159  73.7779
 [6,] 356.750 225.953  79.4531
 [7,] 389.097 273.068  77.8316
 [8,] 396.130 248.455 114.3150
 [9,] 396.833 239.314  60.8060
[10,] 406.678 302.603 112.6940
[11,] 415.819 254.081  90.8036
[12,] 391.207 285.022  90.8036
[13,] 328.318 238.559  99.7218
[14,] 345.502 275.218  89.9928
```

which as you can see does a pretty good job of transforming the second set of landmarks onto the first.
You can also find the inverse of the AffineMatrix using R's built in **solve** function.

```
> solve(AffineMatrix9Dof)
              [,1]        [,2]      [,3]       [,4]
[1,]  1.041870e+00 -0.03569779 0.1822375 -334.94909
[2,]  3.816045e-05  1.04618551 0.2047150 -179.50322
[3,] -2.884093e-01 -0.31072480 1.5879973   89.93745
[4,]  0.000000e+00  0.00000000 0.0000000    1.00000
```

and then check that this maps landmark set 1 -> 2

```
> TransformPoints(landmarks[[1]], solve(AffineMatrix9Dof))
           [,1]      [,2]      [,3]
 [1,]  72.06816 100.42138 49.549280
 [2,]  56.09456  22.08217  8.146006
 [3,]  35.84884 126.95802 78.197776
 [4,] 141.38258 149.44118 -3.419299
 [5,] 132.13752  96.28369  1.637928
 [6,]  43.15121  73.16438 43.009526
 [7,]  74.87517 122.12470 16.465613
 [8,]  89.72991 103.84391 80.020639
 [9,]  81.03731  83.32666 -2.313923
[10,]  98.49118 160.16132 57.579228
[11,] 105.75779 104.91736 35.257971
[12,]  79.01077 137.28645 32.742166
[13,]  16.77248  90.50082 79.479224
[14,]  31.59434 126.86192 47.682712
> landmarks[[2]]
          [,1]     [,2]    [,3]
 [1,]  74.4207  98.2352 50.5358
 [2,]  47.5778  22.4161 10.0879
 [3,]  35.1298 125.9260 71.9297
 [4,] 139.6640 139.8640  0.0000
 [5,] 134.8600  97.4344  7.0000
 [6,]  45.1979  73.4176 45.0000
 [7,]  74.8186 119.8500 16.0000
 [8,]  99.6359 114.2460 77.0000
 [9,]  76.4197  76.2196  0.0000
[10,]  97.6345 161.8790 57.0000
[11,] 105.2400 108.2420 31.0000
[12,]  79.6220 137.4620 34.0000
[13,]  14.7767  89.4288 65.0000
[14,]  34.3904 130.2570 52.0000
```

Saving a transformation
If you want to save a transformation you can either write out the affine matrix eg:

```
write.table(AffineMatrix9Dof,file="~/Desktop/Affine.mat",row.names=F,col.names=F,sep="\t")
```

or you can create an IGS registration file from the registration parameters. First convert the parameters into a list on the correct registration format:

```
igsreg=IGSParamsToIGSRegistration(AffineTransform9Dof)
```

Then either write out a whole registration folder:

```
WriteIGSRegistrationFolder(igsreg,"testaffinereg")
```

or overwrite the registration file of an existing IGS registration folder

```
WriteIGSTypedStr(igsreg,"testaffinereg/registration")
```